THE UNIVERSITY OF MICHIGAN

Technical Report 11

CONCOMP

December 1968

SPECIALIZED SYSTEM SOFTWARE FOR INTERACTING DEC PDP-7 AND IBM 1800 COMPUTERS

R. F. Brender, D. R. Frantz, J. L. Foy, Jr., and T. W. Schuniot

MISSING PAGE
NUMBERS ARE BLANK
AND WERE NOT
FILMED

THE UNIVERSITY OF MICHIGAN

Technical Report 11

SPECIALIZED SYSTEM SOFTWARE FOR INTERACTING DEC PDP-7 AND IBM 1800 COMPUTERS

R.F. Brender
D.R. Frantz
J.L. Foy, Jr.
T.W. Schunior

CONCOMP: Research in Conversational Use of Computers
F.H. Westervelt, Project Director
ORA Project 07449

supported by:

ADVANCED RESEARCH PROJECTS AGENCY
DEPARTMENT OF DEFENSE
WASHINGTON, D.C.

CONTRACT NO. DA-49-083 - OSA-3050
ARPA ORDER NO. 716

administered through:

OFFICE OF RESEARCH ADMINISTRATION ANN ARBOR

December 1968

ABSTRACT

A collection of programs written for interacting DEC PDP-7 and IBM 1800 is described. These programs provide:

1. device support for interaction between 1800 and PDP-7,
2. a serial-by-character logical file system on the 1800 disk (2310) for use by both computers,
3. a file manipulation utility package,
4. a file-oriented text editor running on PDP-7 used for preparing both PDP-7 and 1800 programs,
5. modifications to the assemblers of each computer to read from the logical file system, and
6. a keyboard-oriented debugging package for the 1800.

A (temporary) single-character, full-duplex interface between 1800 and PDP-7 is also described.

TABLE OF CONTENTS

	Page
ABSTRACT	iii
LIST OF FIGURES	ix
LIST OF TABLES	xi
1. INTRODUCTION	1
2. PDP-7 SYSTEM COMPONENTS	7
2.1 LOCG Editor	7
2.1.1 Concatenating Commands	11
2.1.2 Control Characters in Text Mode	11
2.1.3 Exception-handling	12
2.1.4 Switch Options	13
2.1.5 Summary of Editing Commands	14
2.1.6 Appendix A: Rules for Value of "."	21
2.1.7 Appendix B: Special Characters	25
2.1.8 Appendix C: Internal Organization	29
2.2 ML-I	36
2.2.1 Disk Version	36
2.2.2 Counted Continue Command	38
2.2.3 System Information	38
2.3 Assembler	41
2.4 An Interrupt-Compatible DDT	43
2.5 Core Image Program and System Loaders	45
2.5.1 Usage	46
2.5.2 Commands	47
2.5.3 Format	49
2.5.4 Disk Loader	50
2.5.5 Bootstrap	51
3. 1800 LOGICAL FILE SYSTEM	54
3.1 Disk File System: User's Guide	54
3.2 Disk File Utility Program: User's Guide	60
3.3 PDP-7 Interrupt Service: "PDP7"	64
4. DEBUG: A KEYBOARD DEBUGGING PACKAGE FOR IBM 1800	71

TABLE OF CONTENTS (cont'd)

		Page
4.1	Introduction	71
4.2	Arguments	72
4.3	Registers and Commands	73
4.4	Sample Session	77
5.	DISK ASSEMBLER FOR IBM 1800	79
5.1	Reading Source Lines from Disk Using IBM 1800 TSX Assembler	79
5.2	Literal Constants	79
5.3	The Format of Literal Constants	80
5.4	How to Use CDISK, FDISK, and the Modified Assembler	81
6.	1800-PDP-7 INTERFACE	89
6.1	The "Minor" 1800-PDP-7 Interface	89
BIBLIOGRAPHY		93

LIST OF FIGURES

Figure		Page
1	Normal Assembly Operation.....	84
2	Modified Assembly Operation.....	85
3	Operation of LITS within CDISK.....	86
4	Example of Literal Usage: Source Listing....	87
5	Example of Literal Usage: Assembly Listing.....	88

LIST OF TABLES

Table		Page
1	"PDP7" Commands and Services.....	68
2	"PDP7" Error Codes.....	70
3	DEBUG Registers and Commands.....	74

SPECIALIZED SYSTEM SOFTWARE FOR INTERACTING DEC PDP-7 AND IBM 1800 COMPUTERS

R.F. Brender
D.R. Frantz
J.L. Foy, Jr.
T.W. Schunior

1. INTRODUCTION

This is the second of three related reports describing work performed by members of the Logic of Computers Group, a research unit of the Department of Computer and Communication Sciences at The University of Michigan.

The Logic of Computers Group computer facility consists of two,small, general-purpose computers and related peripheral equipment. It is intended to provide a vehicle for heuristic investigation of problems involving large-scale simulations of generalized adaptive systems, including a large class of biologically oriented models.

This report documents those portions of the system software that are largely or completely finished, and that are not likely to undergo further substantial development. It is intended to:

1. serve as a progress and research report describing the capabilities of the current software,
2. serve as a user's manual, and
3. provide enough system information to allow later users to modify or maintain the system.

While these are system types of programs, their development has been necessary to allow for further work. The LOCOSS system for the PDP-7 and the 1800 file system are basic and flexible tools. Descriptions of several other systems components are included for completeness. The particular hardware configuration is summarized at the end of this section.

In general, the TSX system provided by IBM is the basic software nucleus for the 1800. LOCOSS is the basic software nucleus on the PDP-7.

LOCOSS is the Logic of Computers Operating System for the PDP-Seven. It was developed to provide a suitable run-time environment in which to run application programs. It provides buffered, overlapped, and essentially device-independent input/output. A keyboard Command Interpreter provides a number of real-time control services and simple debugging aids. Multiprogramming capabilities are an essential part of the system organization and allow flexible organization of application programs.

LOCOSS, in our estimation, provides unusually flexible capabilities and services on a machine of this size, and requires less than 2K (decimal) of core.

The availability of bulk storage on the 1800 disk via the "minor" 1800-PDP7 interface (in use since April 1968) made it feasible to provide system programs and, perhaps more importantly, user source files, "on-line."

To implement this, a disk file system was developed for the 1800. This system provides variable-length, serial-by-character data files to both 1800 and PDP-7 users. Both symbolic and binary data are kept on-line in this manner. A keyboard utility routine on the 1800 provides simple means to load, dump, list, or copy from or to all 1800 I/O devices and the disk files. Even the approximately tripled listing rate possible with the 1050 printer (15 characters per second, hardware tabs) has been very useful. The pace of program development accelerated greatly as it became possible to be more and more disk-dependent. It was necessary to provide a new text editor because modification of the available one proved impossible. In addition to taking advantage of the device-independent I/O of LOCOSS, the editor provides a couple of string search and replacement commands that are quite useful.

The PDP-7 Assembler was adapted to accept disk file input, although it still punches object code on tape. DDT was made available on-line, and may be loaded and called from LOCOSS. A very powerful macro language, ML-I, was adapted to the disk I/O and made a part of the system. Thus program creation, editing, assembly, debugging, and execution all take place on-line under control of LOCOSS with a minimum of superfluous hard copy generation.

To facilitate debugging of programs for the 1800, a simple keyboard debugging program was devised. The 1800 assembler was modified to read source files from the disk. This made it possible

to use ML-I to process 1800 source code. A symbolic literal preprocessor was built into the disk load phase preceding assembly, adding an important capability to this basic assembler.

While the current interface is sufficiently fast for these human-oriented tasks, it will not suffice for the kind of interactive processing desired for the problem-oriented system. Therefore, the authors designed a high-speed, general-purpose interface (described in a separate report*). It offers flexibility and control substantially beyond current interfacing practice as we know it. The general ideas employed in it should be very useful in other multiple computer systems (as opposed to multiple CPU systems with common memory). Implementation of this interface should be completed by the end of 1968; more complete reports on it will be issued later.

---

* Brender, R. F., and Foy, J. L. Jr., Flexible High-Speed Interface between IBM 1800 and DEC PDP-7 Computers, Technical Report 12, Concomp Project, University of Michigan, Ann Arbor, October 1968.

PDP-7 System Summary

CPU

8K of 1.75 μsec core
18 bits/word
hardware interrupt

Teleprinter (33KSR)

10 char per second

Paper Tape Reader

8-channel
300 char per second

Paper Tape Punch

8-channel
63 char per second

Dataphone (201A)

synchronous
2000 bits per second
connected to switched network

CRT Display (Modified 338)

A display consisting of a DEC 338, less the PDP-8 portion of the 338, is interfaced to the PDP-7. This is locally known as a 337 and is the prototype for the DEC 339. The display operates asynchronously from instruction files in the PDP-7 memory. It provides point, increment, short vector, vector, and character plotting modes, and is capable of branches and sub-routining as well as conditional branches depending on the state of user-controlled switches.

1800 System Summary

CPU (1801C2)

- 16K of 2μsec core
- 16 bits/word + parity and storage protection
- priority interrupt system (12 levels)
- 3 index registers
- 1- and 2-word instruction formats
- 4 data channels

Keyboard-Printer (1816)

- 15 char/sec

Card Read-Punch (1442)

- Read 300 cards per minute
- Punch 60 cards per minute

Disk (2310A1)

- 1 drive
- movable head
- interchangeable cartridges (2315)
- 512,000 words per cartridge

2. PDP-7 SYSTEM COMPONENTS

2.1 LOCG Editor

This section describes a symbolic text editor written for the Logic of Computers Group PDP-7. This implementation assumed 8K core and EAE. It takes maximum advantage of a modified 338 display and the full-duplex teletype on the LOCG PDP-7, but may be run completely from a standard teletype and PDP-7. It is designed to run under the LOCOSS monitor.*

The LOCG editor is designed to provide simple and efficient means to edit symbolic text. Text is organized into lines, which are delimited by carriage returns, and pages, which consist of a number of lines delimited by an end-of-page character. The normal mode of operation is to read into core memory a page of text, to make additions, deletions, insertions, or other changes, and then to write the page out onto an appropriate device (for example, paper tape). A command interpreter accepts user commands and carries out the desired action by invoking appropriate routines.

Every line has an implicit number associated with it which is its position in the page. Any line may be referred to by its line number, and a contiguous block of lines may be referred to by its beginning and ending line numbers. Two special characters may also be used to refer to lines: "/" always has the

* Frantz, D.R., Brender, R.F., and Foy, J.L. Jr., LOCOSS: A Multiprogramming Monitor for the DEC PDP-7, Technical Report 10, Concomp Project, University of Michigan, Ann Arbor, October 1968.

value of the last line of a page; "." has the value of the current line. The current line is defined to be the line most recently referred to in any command. Lines may also be identified relative to these values, for example "/-1" refers to the next to the last line.

All input and output is accomplished with respect to four logical input-output "ports": command source, command sink, text source, and text sink. Any of several devices may be optionally assigned to each of these ports. Thus the text source, for example, may be the tape reader, disk, or dataphone. Assignment is controlled by the user with an appropriate command.

The command source is almost always assigned to the keyboard of the PDP-7 teletype. All commands are taken from there. The command sink is the device on which the editor responds to the user. This is normally assigned to the display. The text to be edited is read from the text source, and the corrected text is written on the text sink.

All functional descriptions of command actions are given in terms of the port used rather than any particular device. In this implementation, the teletype is also called the "master device." Certain error conditions will have the effect of re-establishing the master device as the command source allowing the user to recover control.

Either text or commands may be entered from the command source. A command such as Insert will, for example, cause the

editor to interpret the following lines as text for insertion into the page. Thus the editor may be considered to be in one of two "modes": expecting a command or expecting text to satisfy a previous command. When the editor is in command mode, it signals its readiness to accept a command by issuing an "*" character to the command sink. When text is expected, no prompting character is given. The user may leave text mode by typing an EOT character. EOT is the normal means for terminating additions, insertions, changes, etc.

The general organization of the command structure is similar to that used in the DEC Symbolic Editor. However, the generalized input/output capabilities, more flexible command structure, and the addition of several powerful commands add immensely to the utility of this editor. See the DEC Editor write-up for extensive examples.

The command syntax is as follows:

(where { } indicates an option
[ ] indicates alternatives)

<ARG>{,<ARG>}<CMD>{SPACE<ARG>}[SPACE or CR]

For example, "8PAG" or ".+2,/L" or "A" are valid commands.

An <ARG> has the following syntax:

```
    <ARG> :: = <VALUE> | <ARG> <OP> <VALUE>
  <VALUE> :: =  . | / |<NUMBER>
     <OP> :: =  + | -
 <NUMBER> :: = <DIGIT> | <NUMBER> <DIGIT>
  <DIGIT> :: = 0|1|2|3|4|5|6|7|8|9
```

A <CMD> consists of one, two, or three letters. A sequence of commands may be placed on one line separated by spaces. Any error in processing any command will terminate the entire line.

Many commands may optionally apply to all of the current page, a single line of the current page, or to a given sequence or block of lines of the current page. Other commands may be used in only one or two of these three cases. If one of these commands is given without any arguments, then the whole page is assumed as the domain of the command. If a single argument precedes the command, then only that line is the domain of the command. If two arguments are given, then the block of lines is the domain of the command. For example, "L" will cause the entire page to be listed on the command sink, "3L" will cause the third line only to be listed on the command sink, and "1,6L" will cause the first six lines of the page to be listed. The commands whose arguments are interpreted in this way are described in Appendix A. Appendix B describes the commands for which interpretation of the arguments is different. Appendix C describes those commands that require a third argument.

2.1.1 Concatenating Commands

Because commands can be terminated by a SPACE, more than one command can be placed on a line. These are read and performed in order. For example

```
*2CPY SPACE 1SKP SPACE 3 CPY SPACE EOF CR
*
```

would cause the entire sequence of commands to be performed before prompting for the next command input. An error on any command, however, will terminate the entire line and prompt for a new command.

2.1.2 Control Characters in Text Mode

Any of the following characters will cause the editor to terminate a line and return to command mode when encountered during adding text to the buffer: EOT(204), EOP(214), EOF(023 or 223). Normally only EOT is used from the keyboard for this purpose.

EOT will return the editor to command mode and terminate the current line, if any, with a CR. Hence the following sequence of commands is equivalent.

```
*A
THIS IS ADDED TEXT CR
EOT CR
*
```

and

```
*A
THIS IS ADDED TEXT EOT CR
*
```

A blank line will not be formed by line three of the first example.

In view of the section on concatenating commands, the above can also be performed by:

```
*A SPACE THIS IS ADDED TEXT EOT CR
*
```

One additional control character will be recognized in text mode. This is WRU (Control E). It is intended primarily for preparing tapes on an off-line teletype, but will function exactly the same on-line. WRU means delete this entire line up to this character and replace it with the following text. If CR follows, then it is also deleted. Otherwise the next character begins a normal line. When a tape containing WRUs is read by the editor, via "R", the indicated lines will be automatically deleted.

2.1.3 Exception-handling

A number of events will cause termination or modification of the currently executing command. The simplest of these are the command errors, i.e., mistakes by the user in giving a command to the editor. The editor responds with a question mark to the command sink, a bell to the master sink (teletype), clears the command line, and asks for a new command. For example,

arguments out of range, illegal command characters, or an incorrect number of arguments will get this response. In general, nothing will have been changed as a result of the error.

End-of-physical-record indications may be encountered when reading the text source. The effect is to terminate the current (possibly partial) line in normal fashion, give an error comment, terminate the read, and return to command mode. Further stacked commands will not be processed.

When the core buffer is nearly full, the editor will terminate any command seeking to add to the buffer, e.g., R, I, A, S, with the comment CORE ALMOST FULL, and return to command mode. The buffer may still be added to until it is physically full. After that, any command adding to the buffer is treated as a command error. About 100 (decimal) characters may be added to the buffer after the initial warning is given Note that this warning allows one line to be added, then terminates the command. The message CORE FULL indicates that nothing was added to the buffer.

2.1.4 Switch Options

Switch 0 may be used to halt output operations after they are already in progress. Putting switch 0 up internally forces a command error condition and returns the editor to command mode. Because of buffered I/O, output may not stop immediately. In this case, the user should wait for the prompting character, put switch 0 down, and resume.

As a more extreme measure, the user can do a hardware STOP and start over; before restarting, he should raise switch 2. This will prevent the editor from killing the buffer when initializing.

2.1.5 Summary of Editing Commands

Group 1.

This group of commands includes most of the text manipulation commands. They all interpret the arguments given (if any) as specifying the domain of the command.

Command	Acronym	Arguments	Effect
A	Append	0	Insert text from command source at the end of the page.
B	Back-up	0	List previous line. Equivalent to ".-1L".
C	Change	1,2	Delete the given line(s) and replace with the following text from the command source.
D	Delete	1,2	Delete the given line(s).
I	Insert	1	Insert the text that follows after the given line.
IB	Insert Before	1	Insert the text that follows before the given line.
L	List	0,1,2	List given lines on command sink.
W	Write	0,1,2	Write the given lines on the text sink. If no arguments are given, then an end-of-page character automatically follows the last line.
WK	Write and Kill	0	Write entire buffer followed by end-of-page, then kill buffer.
WKR	Write, Kill, and Read	0	WK followed by reading next text page from text source.

Group 2

For these commands the argument, if present, is interpreted as a function of the command.

Command	Acronym	Arguments	Effect
CDS	Clear Display	0	Clear the display.
CPY	Copy	0	Copy one page from text source to text sink. Error if core buffer not empty.
CPY	Copy	1	Copy the given number of pages from text source to text sink. Error if core buffer not empty.
E	Evaluate	1	Print the value of the argument on the command sink. Commonly used with "." or "/".
EOP	End-of-Page	0	Write an end-of-page character on the text sink.
EOF	End-of-File	0	Write an end-of-file character on the text sink. Should be used preceding physical end of paper tape.
K	Kill	0	Empty entire buffer.
N	Next	0	List next line. Equivalent to ".+1L".
N	Next	1	List next given number of lines. Equivalent to ".+1,.+<ARG>L".
PAG	Page Copy	0	List the current core buffer on master device in page form, i.e., with sufficient extra blank lines to occupy physically 11 inches on teleprinter. Error if buffer empty.
PAG	Page Copy	1	List the given number of pages from the text source on master device in page form. Error if current buffer not empty.

Group 2 (continued)

Command	Acronym	Arguments	Effect
R	Read	0	Read one page from text source and add to end of buffer.
R	Read	1	Read given number of lines from text source and add to end of buffer. Will be terminated by an end-of-page or end-of-file character efen if count is not satisfied.
RCL	Reader Clear	0	Stop high-speed reader and clear reader buffer.
SKP	Skip	0	Skip one page of text from text source. Current buffer not affected.
SKP	Skip	1	Skip given number of pages from text source.

Group 3.

The reamining commands need some elaboration. Most require a "third" argument which follows the command. This argument is always separated from the command by exactly one space.

Command	Acronym	Argument
DOC	Document	1

This command is designed to assist in adding documentation—comments—to already existing assembly language code. The argument specifies an initial line in the buffer. That line is printed on the command sink, then the user indicates what action is to be taken. A carriage return indicates to leave the line as is and go on to the next line. Any text before the carriage return will be appended to the end of the line. Then

the next line will be listed, and so on. This mode may be terminated by EOT as usual.

WRU may be used in this mode to replace an entire line. For example, after the line is listed, enter: WRU NEW TEXT LINE CR. Be careful in using the WRU in this manner. Note that:

WRU CR
NEW TEXT LINE CR

will have the same effect as WRU NEW TEXT LINE CR, i.e., the editor will expect a replacement line to follow after the WRU CR pair.

WRU CR CR will replace the deleted line with a blank line. Also note that WRU EOT CR or WRU CR EOT CR will result in a null replacement line (i.e., the line being deleted and termination of the DOC command).

Command	Acronym	Arguments
M	Move	1,2+1

Move the lines given by the first arguments and insert them after the last argument. 8,10M 15 would insert lines 8 to 10 inclusive after 15. The syntax of the third argument is the same as for the first two. The editor responds by listing two lines beginning at the third argument.

Command	Acronym	Arguments
F	Find	0,1,2+1

The third argument has the form <delimiter> <string> <delimiter>. The <delimiter> can be any character except a carriage return. The <string> is any sequence of characters except carriage return or the delimiter. For example, /A STRING #/ would indicate a text string of ten characters.

With no initial arguments, the core buffer is scanned for the first occurrence of the given string. When found, the containing line is listed on the command sink. The value of "." is set to that line. If the scan fails, the value of "." is unchanged. With one initial argument, the scan begins at the given line and proceeds to the end of the buffer. With two initial arguments, the indicated block is scanned. (Note that a pattern cannot extend across two lines.)

Command	Acronym	Arguments
CF	Continue Find	0

This command allows the user to search for successive occurrences of a string. When issued after a Find command, the scan is resumed after the last line successfully matched and looks for the same string as previously given in the Find. Any commands except the S command may be interposed between an F and CF or between two CF commands without destroying the memory of the desired string.

The CF command will search to the end of the page if no successful match occurs. If a match occurs, action is the same as for the F command.

Command	Acronym	Arguments
S	Substitute	0,1,2+1

The third argument has the form <delimiter> <string> <delimiter> <string> <delimiter>. With no arguments, the entire buffer is searched for an occurrence of the first string. Where found it is replaced by the second string. Only the first occurrence in a line is replaced. The line numbers of all modified lines are listed on the command sink. With one argument, only the given line is scanned. With two initial arguments, the given block is scanned. The dot has the value of the last line modified. If no line is changed, the value of dot is unaffected.

Command	Acronym	Arguments
DIO	Define I/O	0+1

This command allows the user to assign a device to an editor port. The argument has this syntax: (Note that these commands must end with a CR.)

```
<ARG'> :: = <DEV.SUBCOM> CR
<DEV.SUBCMD> :: = <PORT> = <DEVICE>
<PORT> :: = CSC|CSK|TSC|TSK
<DEVICE> :: = TTY|TAP|DSK|DPH
```

TTY refers to the keyboard when used on a source and to the teleprinter when used as a sink. TAP refers to the tape reader

when used as a source and the tape punch when used as a sink. For example, to assign the display to the command sink, enter "DIO CSK = DSP."

At present the allowable assignments are the following:

Command Source: CSC = TTY or TAP

Command Sink: CSK = DSP or TTY

Text Source: TSC = DSK or TTY or TAP or DPH

Text Sink: TSK = DSK or TTY or TAP or DSK or DPH

The underlined device is the device initially assigned when the editor is loaded. (Note that when the display is the command sink the teletype should be full duplex.)

Command	Acronym	Arguments
OPI	Open Input File	0+1
OPO	Open Output File	0+1
CLI	Close Input File	0
CLO	Close Output File	0
CRE	Create a File	0+1
DES	Destroy a File	0+1
CLR	Clear and Reset	0

This group of commands is concerned with manipulating the 1800 disk file system. Several of the commands require a following argument which is a decimal logical file number. Error or failure conditions are reported by the comment "DISK NAK #" where # is the error code. Consult Section 3.3, PDP-7 Interrupt Service: "PDP7," for details.

APPENDIX A

RULES FOR VALUE OF "."

APPENDIX A

RULES FOR VALUE OF "."

Operation	Resulting Value of Dot
Successful Search (F of S)	(Last) line found.
Unsuccessful Search	Unchanged.
Insertion	Last line inserted.
Deletion	Line preceding first deleted line.
List or Write	Last line listed or written.
Move	First line of inserted block.

APPENDIX B

SPECIAL CHARACTERS

APPENDIX B

SPECIAL CHARACTERS

The following characters are interpreted by the Editor in special ways. None of these characters may be entered into the buffer.

Name	Octal Code	Function
EOF	223,023	End-of-record mark.
EOP	214	End-of-page. Separate parts of text.
EOT	204	End-of-text. Terminates text entry.
WRU	205	Line delete. Deletes line currently being entered.
LF	212	Ignored.
NULL	000	Ignored.
RUBOUT	377	Ignored.

APPENDIX C

INTERNAL ORGANIZATION

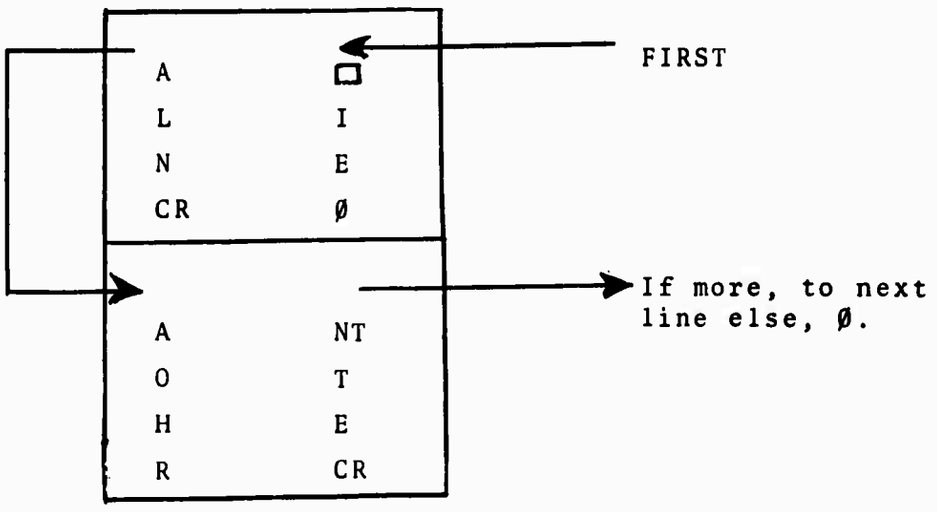
BLANK PAGE

APPENDIX C

INTERNAL ORGANIZATION

LOGIC EDITOR: SYSTEMS INFORMATION

The basic data structure used in the editor is a simple linked list of variable-length blocks. Each block represents one line of text and is terminated by a carriage return (octal 215). The position of that line on the list is the number used when referring to a particular line. The first word of a line block is a pointer to the first word of the next line block. The last line block has a pointer of zero. Each line is stored two characters per word (8 bits per character) right-justified in a computer word with the left (high-order) character logically the predecessor of the right character. Note that the terminating carriage return may occur in either character position.



Several variables provide all the needed information to access and modify this structure. FIRST is a pointer to the first line in the buffer. THISN is the number of the current line in the buffer and corresponds with the dot "." used in referring to lines. LASTN is the number of lines in the buffer (and hence the number of the last line) and corresponds with the slash "/".

Free storage is acquired sequentially upward in core. The next available location is BUFNXT. The limits of the buffer are given by BUFBEG and BUFEND. The variable BUFALF gives the point at which the BUFFER ALMOST FULL message is given. Lines are deleted simply by removing them from the link sequence. No attempt is made to recover or keep track of the storage thus released. Core is recovered only by the Kill command which resets all pointers to correspond to an empty buffer with free storage beginning at BUFBEG.

All commands are one, two, or three characters possibly preceded by zero, one, or two parameters. The command processor collects the parameters and commands and then searches for the name in the command dispatch table. This table is a four-entry table whose first entry is the command code in six-bit trimmed ASCII left-justified with up to three characters in the word. The remaining entries are the routines to be called to process the command if given with zero, one, or two arguments respectively. The decision of which routine to call is made by the command interpreter.

If arguments are given, they are stored in locations ARG1 and ARG2. If ARG2 or both ARG1 and ARG2 are missing then they

have the value minus one. The interpreter also checks that (where appropriate) arguments are within legal ranges, i.e., correspond to lines within the buffer.

Where a given combination of arguments is illegal, the command table has a call to the routine CER (command error) which informs the user that he has goofed.

Commands which involve an argument following the command name perform their own analysis of the argument, including checking that it is separated from the command by a space and not a carriage return.

The variable CERSAV controls whether the command processor will prompt the user for a new command. Basically it contains the character that terminated the last command. If that was a CR, then a prompt is given and a new input line is sought from the user. Otherwise more input is available and no prompt is given. (Prompting is controlled by the editor rather than by the system, because the prompt must be given on the command sink rather than always on the teleprinter.) Routines with third arguments are responsible for maintaining the correct contents of this variable.

The heart of the editor is a rather involved pair of routines called INS (insert a line) and PACK (accumulate a line in packed form). These are governed by several parameters which must be established before the routines are called:

SC - the routine to call which will return the next character to enter the line. This may be an external device-support routine or an internal character-manipulation routine.

ARG1 - The new line will be inserted after this line.

TERM - The address of a routine which will detect when the insertion process is to terminate. The occurrence of one of a number of special characters or of a given number of carriage returns are two conditions currently used. When the condition is detected, the variable TFLAG is set non-zero. This word is tested at various points to determine the course of action.

The PACK routine is called by INS and will return only when it has accumulated a full line and/or has detected a termination condition. The new line is linked into the rest of the structure only after the return to INS, thus minimizing the possibility that hardware or user faults will destroy the integrity of the current buffer load. The PACK routine also protects the integrity of the buffer by converting all terminating characters to a carriage return which is what is actually stored in the buffer. This is the reason a text line may be terminated by EOT from the keyboard, for example. An exception to this is the case where the terminating character is the first character of a line, in which case no line is entered into the buffer.

Before returning a complete line to the INS routine, PACK tests whether the value of BUFALF has been exceeded. If so, it prints the message BUFFER ALMOST FULL, sets the termination flag, and returns to INS. On each pair of characters entered,

PACK checks to see if BUFEND has been exceeded. If so, it types the message BUFFER FULL, stops accumulating the line, sets the termination flag, and returns to INS as though no line had been received at all. A message is also printed when an end-of-file terminates the input, and the counters used in counted reads are also reset so that no more input will be sought by higher-level calling routines.

The WRU conventions are also implemented in this routine. When a WRU is encountered, the PACK routine returns to its own start and begins to accumulate a new line all over again as though it were just called.

All input/output is handled through the four locations: CSC (command source), CSK (command sink), TSC (text source), and TSK (text sink). The routines to use are stored in these locations, and all parts of the editor use the appropriate port. Special properties of the devices are handled by the device support routines called via these ports.

2.2 ML-I

2.2.1 Disk Version

A version of Macro Language I by P.J. Brown of the University of Cambridge is now available on the 1800 disk. Modifications have been made to allow it to read and/or write from disk files in addition to paper tape input-output. All input and output is assumed to be in ASCII; that is, translation to and from Titan Flexowriter code has been deleted. Several additional commands have been added to allow user-control of the needed disk files.

ML-I is loaded in normal fashion by LOCOSS.* ML-I overlays all of LOCOSS and runs as a separate and independent system. When finished, the user types QT (quit), and ML-I will automatically reload LOCOSS.

The following commands have been deleted from the disk version of ML-I: IA, IF, OA, OF, and FP. (The last, while recognized, is now equivalent to FL.)

The following commands have been added. The symbol "#" is used to represent a decimal disk-file number.

1. IR - Input Reader. Input is from the tape reader in ASCII.
2. ID# - Input Disk. The specified disk file is opened and used as the input source.

---

* Frantz, D.R., Brender, R.F., and Foy, J.L. Jr., LOCOSS: A Multiprogramming Monitor for the DEC PDP-7, Technical Report 10, Concomp Project, University of Michigan, Ann Arbor, October 1968.

3. IDC - Input Disk at Current file. After a disk file is opened and partially read, the input may be switched to the tape reader without affecting the disk. Input may be restored to the "current" file by IDC, which simply causes the next input to be sought from the disk.

4. ICL - Input Close. Close the current disk input file.

5. OP - Output Punch. Output is to the tape punch in ASCII.

6. OD# - Output Disk. The designated disk file is opened for output.

7. ODC - Output Disk at Current file. As with IDC, output may be switched away (or off via ON, Output None) and back without affecting the disk.

8. OCL - Output Close. Close the current disk output file. It is important to do this lest a part of the output be lost!

9. QT - Quit. Quit and reload LOCOSS.

The remaining commands, in particular S, T, ON, and CK, have their previous effects.

Errors in using the disk routines are reported by the comment "NK#" where # is the error code in decimal. An end-of-file is treated as a normal stop code and a translation error (between ASCII and EBCDIC) is reported, but the process continues. Other errors during reading or writing disk files are considered cause to abort a process.

2.2.2 Counted Continue Command

The two commands "CN" and "CT" have been augmented to provide for an optional count to follow the command. The effect of this count is to allow a given number of text pages to be processed before ML-I requests a new command. The syntax is "CT" or "CN" optionally followed by a decimal count followed by a carriage return. If the count is absent, a count of 1 will be assumed. The carriage return is necessary in either case. An invalid command or count will invoke a complaint, after which a new command will be sought.

The "T" (title) option applies only to the first page processed. Additional pages are treated like the CN command except that the first line is not typed out. The stop message at the end of each page is still given, though in shortened form.

Most errors will revoke the count and allow the user to take corrective action,if possible.

2.2.3 Systems Information

This section describes the procedure by which the disk version may be generated. Because ML-I is a system which overlaps and is independent of LOCOSS, the facilities of LOCOSS are not available for generating ML-I. The following steps should be followed carefully.

ML-I as received from P.J. Brown consists of eleven tapes identified as Part 1, Tapes 1 through 5, and Part 2, Tapes 1 through 6, all dated May 1967. These may be assembled using Brown's directions to produce the standard version of ML-I.

The disk version consists of nine tapes identified as Part 1, Tapes 1 through 3, and the same Part 2 tapes as above. The modifications are all in Part 1.

The instructions to the user are as follows: Assemble the tapes together with the DEC assembler. Suppress punching of DDT symbols since the tape is unwieldy with them, and both the symbols and ML-I do not fit in core with DDT anyway. Load the tape into core with the LOAD$ command of DDT. Take the address of first free core typed out by DDT and add at least 14_8 to it. Use this value to replace the two occurrences of STSTAK which may be located by the WORD$ command with 12000 as parameter. Larger values of STSTAK may be used if core space is needed to enter patches with DDT.

Next enter the high core version of the loader (with self-contained disk routines) into the PDP-7 via hardware read-in beginning at 17600. What is now in core is a complete version of ML-I together with a section for punching itself out after the manner of the standard ML-I.

ML-I will write itself out either to paper tape or directly onto the 1800 disk. The code for doing this is located at about 13000_8 and is not located in the regions that are written out. The transfer to these routines is modified before writing so that no attempt can be made to execute this code from the version of ML-I that is written. In short, the FP command becomes identical to the FL command.

ML-I writes itself in PDP-9 absolute binary tape format as described in the PDP-9 literature. This is the standard format for all LOCOSS system binary object files. No loader is provided on the written output, and none is needed on the disk.

Start ML-I at 22_8. Type the command FO. If the output is to be written on the disk, then open the appropriate disk file with the OD# command. (Note that while the OD# command opens a file with translation on, the self-punching routines will close and reopen that file without translation. The file will be closed when finished.) Next enter the command FP. That's all there is to it. It should go without a hitch.

Note that the writing takes place with interrupts off, and hence the message FREE STORAGE...will be suspended at about the third or fourth character until the operation finishes. Disk errors cause a halt. Take action as appropriate.

QT will invoke LOCOSS, and attempted loading of the just-created file will quickly test if the operation succeeded.

The input-output options are controlled by four words as follows:

IDISK - number of currently open input disk file. Zero indicates no file is open. IDISK is set by ID# and ICL commands, and also by end-of-file condition on disk read.

ODISK - number of currently open output disk file. Zero indicates no file is open. ODISK is set by OD# and OCL commands.

DRSW - Disk-Reader Switch. Zero indicates input comes from the reader, while nonzero indicates input from the disk. DRSW is set by ID#, IR, and IDC.

DPSW - Disk-Punch Switch. Zero indicates output is to punch, and nonzero indicates to disk. DRSW is set by OD#, OP, and ODC commands.

The convention on character disk files is that carriage returns are retained while linefeeds are suppressed. ML-I on the other hand suppresses carriage return and recognizes and generates line feeds. Hence the device support routines DSKGT (input) and DSKPT (output) convert from one of these conventions to the other.

ML-I uses the disk routines from high core that are part of the loading routine. The top of ML-I free storage is thus determined by the bottom limit of those routines. The current value is 17577.

2.3 Assembler

This describes the use of an interim assembler for the PDP-7. Source files may be read by this assembler from 1800 logical files or paper tape in any order. Output is always to the paper tape punch in standrad F.F. Format.

The assembler may be loaded from the disk via LOCOSS which remains in core and intact. However, only the disk routines are used by the assembler. Instructions to the user are as follows:

After DAS is loaded, set the address switches to 4400 (octal) and enter in AC switches 10-17 the logical file to be read. Zero

indicates the tape reader, and nonzero specifies a disk file number. (Note that the old switch 10 and 11 options are deleted.) Assembly proceeds in two phases: reading of source information, and completion and symbol table type-out. At the halt at the end of a paper tape (AC = 777777, MQ = 0) or a disk file (AC = MQ = 707070), phase one may be continued by entering the desired file number in the AC switches, then pressing start. DAS halts when finished.

When reading from the disk, PAUSE and START pseudo-ops will not stop the source input. However, the other effects of these commands with respect to loading are carried out. Source reading stops at the physical end-of-file with 707070 (octal) in both AC and MQ. Disk error halts will have the disk error code in the AC and 0 in the MQ.

Disk and paper tape source may be interlaced in an assembly.

Display and LOCOSS symbol definitions are available in disk files in source format and may be included in the assembler tables by reading as the first source tape(s). These symbols will not appear in the symbol table. It is recommended that switches 0 and 2 be raised when reading these files to prevent these symbol definitions from being punched. Switches 0 and 3 may be used to select the correct title for the output tape.

When this assembler is loaded from paper tape, it assumes that all input will be from paper tape, and AC switches 10-17 are ignored except for symbol table options. When loaded in

this manner, operation is the same as the DEC version except:

1. Switch 10, 11, and 14 options are deleted,
2. Use 4400 (octal) rather than 22 (octal),
3. The halt between phases 2 and 3 has been deleted.

A symbol table print-out can be forced most any time by starting at 4401 (octal).

Important: It is necessary to load this assembler fresh each time a new assembly is started.

2.4 An Interrupt-Compatible DDT

This section describes the use of a modified DDT with programs using the "program interrupt" hardware of the PDP-7.

DEC Debugging Tape (DDT) and user programs are loaded in the normal manner. The restart entry remains at 16000_8 and forces interrupts off. DDT may be used exactly like the original DDT without knowledge of these modifications.

The modifications to DDT are concerned with facilitating control of the "program interrupt" as control moves between DDT and user programs and back. The following commands (which remain in effect until the next is given) are provided:

STIOF$ - leaves program interrupt off when exiting to user program. (This is the default condition.)

STION$ - turns program interrupt on when exiting to user program.

STIOC$ - turns program interrupt on when exiting to user program if it was on at the last breakpoint entry.

The latter is particularly useful when continuing a program with "!". Note that a counted continue will have the desired effect.

commands are always executed with interrupts off.

The value of SAVIOS may be determined by examining register SAVIO$.

Interrupts may be easily "lost" since DDT runs with interrupts off and senses device flags directly. This is particularly true of the teleprinter flag since the printer is immediately used by DDT. DDT may be instructed to monitor the printer flag by the command TTY$. This feature is revoked by NOTTY$. The teleprinter flag may be examined via TTYF$. Zero means down and one up. This feature is implemented as follows:

After turning interrupts off on a breakpoint entry, DDT loops approximately 140 milliseconds to determine if a character is being printed. The teleprinter flag is tested at the end of this loop and the state saved. When continuing with "!" or "'", DDT prints a null character just before transferring to the user program if the teleprinter flag was up on entry. This will result in a teleprinter flag coming up after reaching the user program. This flag may be prevented by storing a zero in location TTYF$ Conversely, this flag may be forced on by putting one in TTYF$.

A symbol table consisting of the basic commands and EAE symbols, except IOTs, is loaded automatically. Two additional tables are not loaded: IOTs and a restricted display set. The

IOTs may be loaded by TABLE$ and bypassed by LOAD$. Similarly for the display symbols. Thus either or both may be loaded if desired. This gives greater flexibility in keeping the symbol table small.

Print-out of EAE, OPR, IOT, and LAW instructions has been modified. If an exact equivalent symbol is not found, then the generic type and octal remainder are printed.

2.5 Core Image Program and System Loaders

The CORE IMAGE PROGRAM (CIP) produces and summarizes core-image-formatted information on the disk or on tape, under control from the keyboard. The program runs as a complete package and is available in three versions. The low-core version occupies locations 22-2200 (octal), overlaying only LOCOSS so that all of user core may be dumped. The mid-core version occupies 10000-12200 and the high-core version occupies locations 15600-17777, so that most pre-LOCOSS programs may be "imaged." By judicious bouncing around it is thus possible to put all of core, except the loader area (17600-17777), into core image format for loading from the disk.

All three versions are available on core image tapes in the program library, and the low-core version is also available to be loaded from file six (6) on the disk.

2.5.1 Usage

Once the program to be imaged is in core, the user must load the appropriate version of the CIP from tape or from the disk. The CIP is self-starting, but, if necessary, may be restarted at either 22,10000, or 15600, appropriately. After identifying itself, the CIP immediately issues a disk "CLEAR" command so that all outstanding input and output files will be closed. An error on this disk call will produce the printed message CAN'T CLEAR DISK and the error number, in octal. If this occurs, the 1800 is probably dead, so that further operation will entail resuscitating it or curtailing one's activities to paper tape.

When the CIP is ready for input, it will type a question mark (?) and will accept a command line. The user has available the same input line-editing features as in LOCOSS. A summary of commands is given below. Only the first two letters of each command need be given. For those accepting numeric parameters (octal or decimal), all alphabetic and special characters after the first two characters and until the first digit are ignored. A number is terminated by any non-digit (including space). An unrecognizable command or one lacking sufficient parameters will be greeted with WHAT? and ignored. In general, file numbers are decimal and core addresses are octal. Whenever the user addresses file zero, by convention he refers to the tape reader or punch.

A disk error during the execution of any command will result in printing DISK ERROR and the error number in (octal). A disk CLEAR command will be issued and the CIP will start over. When the user is finished with the CIP he may reload LOCOSS by giving the command END.

2.5.2 Commands

CREATE N. Create file N, where N is a decimal number.

OPEN FILE N. Open file N for output where N is a decimal number. This must be the first command issued in creating a core image file. If N is zero, the hardware read-in mode absolute tape loader will be punched on tape and all further output commands will refer to the punch and not the disk. The tape so produced will be self-loading.

TEXT. Write the text (between the first blank following the command and the carriage return) onto the output device for documentation purposes. The loader will type this information if the program is loaded from the disk, and the information will be read by the DUMP command but it will be ignored by the tape loader. TEXT commands may be given at any time before the file is closed by CLOSE, and each one writes a line into the file.

BLOCK N M. Write the block of core locations from N to M (both octal), inclusive, onto the output device. M must be greater than or equal to N . BLOCK should be repeated for as many contiguous blocks of core as are to be written.

START N. Write a block onto the output device which will cause execution to begin with location N (octal) when the loader reaches this block in the loading process. This produces the same effect as the address N in the location field of the "START" statement in the PDP-7 assembler.

PAUSE. Write a block onto the output device which will cause the tape loader to halt after the program has been loaded, or the disk loader to return to the caller.*

CLOSE. Close the output file, give the count of characters and disk sectors written (in decimal), and restart the core image program.

DUMP N. File N (decimal) is opened and its contents as a core image load are summarized as text, binary blocks, and start or pause blocks. File zero by convention is the tape reader and, if specified, the tape should be positioned after the tape loader. The program will not recognize the end of the tape, so that it will be necessary to restart it after the dump is completed. Indication is given if a checksum error is detected. Information dumped by the program is preceded by a right angle-bracket (>), while text from the file is printed with no prefix character. The number of characters and sectors read is also given (in decimal).

PUNCH N. Punch file N (decimal) on paper tape preceded by the hardware read-in mode loader. The tape so punched can be read into core by setting the address

---

* See Section 2.4 General Service Routines, in LOCOSS: A Multiprogramming Monitor for the DEC PDP-7.

switches to 17720 and pressing READ-IN.

END. Finish the core image program and reload the LOCOSS file.

2.5.3 Format

The core image format is that of the PDP-9 absolute binary loader as stated in MACRO-9 Assembler: Programmer's Reference Manual, DEC-9A-AM9A-D and recapitulated faithfully below.

The information stored on the disk is the direct image, character by character, of the paper tape version of the program. All words are in paper tape reader binary format. See pages 76 to 77 of the PDP-7 User's Manual for the details.

Information is on the tape in blocks of contiguous binary locations, and each block is preceded by a three-word header:

1. Bit 0: = 0 means binary block header.
   Bits 1-17: = Starting address of block.
2. Twos complement of number of words in block.
3. Checksum: twos complement of the sum of all the words in the block including words one and two of the header.

The following n (=word count) words of the block are loaded into the block starting address and successively higher locations in core.

A start block consists of two words:

1. Bit 0 : = 1 means start block.

   Bits 1-17: = all ones if no starting address, i.e., halt after loading.

   = otherwise, the starting address for the program.

2. Dummy word for reader buffering.

The text in the files or on tape is stored as ASCII with the high-order bit zero so that it is not read as binary by the tape reader or by the disk read routines.

2.5.4 Disk Loader

The disk loader will load "Core Image" files from the IBM 1800 disk. As such, it contains a copy of two of the standard disk communication routines, and has provisions for handling disk errors and other possible loading terminations. The disk loader is in file three (3) in "bootstrap" form(see below) and is loaded from the disk fresh when the LOCOSS bootstrap procedure is followed and when the LOCOSS subroutine LOAD is called either from a program or in the CLI response to the LOD command. The loading procedure does not destroy the loader so that it may be used again without itself being reloaded from the disk.

Locations 17772 to 17777 are a local "transfer vector" or "communications area" for the loader. Some of these locations contain parameters for the loader and instructions for the loading sequence, so that setting these locations constitutes the calling sequence for the loader.

17772 - Disk error location. Any loading error detected by the disk routines will cause control to be transferred to this location. Typically this will contain a jump to a user routine or a halt instruction.

17773 - Pause location. If the file being loaded ends with a "Pause" block, control will be transferred to this location.

17774 - Number of the file to be loaded. Only bits 10-17 of the word are considered (i.e., a LAW instruction may be used).

17775 - Address of the entry point for DSKERR.

17776 - Address of the entry point for DISK. Both of these routines have the same calling sequences as their counterparts in the resident part of LOCOSS, and may be used as described in the LOCOSS manual when they are in core and LOCOSS is not.

17777 - Jump to the start of the loading process. Thus the normal method of starting the loader is to set locations 17772-17774 and to JMP to 17777.

The format of a core image file is given following the description of the Core Image Program.

2.5.5 Bootstrap*

The disk loader itself is stored on the disk in file three (3) in a format which is designed to require only a minimal program for bringing it into the PDP-7. Because the bootstrap load also does not follow the normal procedure for

* This section is intended for system maintenance purposes, not properly being a user-available function. It is included here for completeness sake in the loader description.

obtaining information from the disk, great care must be exercised in using it since it is very easy to hang-up the IBM 1800 otherwise.

To initiate a bootstrap load, the PDP-7 presents the 1800 with the bootstrap op code and the file number to be loaded. The 1800 then returns the first character of the file, with the acknowledment bits set. The 1800 service routine remains in its interrupt state, feeding the characters from the file to the PDP-7 as fast as the PDP-7 will take them. Note that the normal request-acknowledgment procedure is followed only for the first character of the file. The rest of the characters in the file are piped indiscriminately to the PDP-7 without its having to ask for them, until the end of the file is reached, at which time the 1800 interrupt service routine exits from interrupt status. Thus, if the PDP-7 is not prepared to handle the whole file and, for example, it stops reading, the 1800 will simply wait for the 7 to finish and not return to its own processing. During this time the 1800 will not recognize anything written at it, so that it is impossible to terminate the bootstrap in any other way than that intended.

The only way in which the bootstrap load will not be started is for the bootstrap file number request to be in error. The only files from which bootstraps are allowed are numbers 1-10 (decimal). A bootstrap does not affect the "open" status of any file serviced by the 1800 service routine.

The bootstrap procedure is obviously meant only for debugged system programs. (The preceding specification of program type is hopefully redundant.)

The format of a bootstrap file is as follows:

1. The first character of the bootstrap file 3 is the number of the core image file for LOCOSS, so that when the disk loader has been bootstrapped in, it may start loading LOCOSS. In other bootstrap files, the first character may be used as desired.
2. The remaining characters in the file are combined, three at a time, to form eighteen-bit PDP-7 words. The high-order two bits of every character (except the third last) are zeroes. The low-order six bits of the first character, then, are the high-order bits of the word, etc.
3. The first word thus formed is treated as an address which is the base of an area into which the remaining words will be loaded. The second word will be stored in the location addressed by the base. The next word will be stored in base + 1, etc.
4. The third last character in the file has the second high-order bit (i.e., bit "100") set. This bit being set indicates that the character starts the last word in the file. This word is not stored at the end of the block but is executed. Thus, it is normally a no-op, halt, or jump instruction.

Currently the only files in bootstrap format are three (the disk loader) and five (the RIM and FF loaders).

3. 1800 LOGICAL FILE SYSTEM

3.1 Disk File System: User's Guide

The disk file system for the IBM 1800 is a collection of subroutines, designed for use under TSX Version 3, by which a user can create and maintain a set of logical files on the 2310 disk. Each file behaves as a serial character source and/or sink of indefinite length. Files are identified by number; the maximum allowed number of logically distinct files is somewhat arbitrary (depending mainly on the amount of disk storage available), but is currently fixed at 150. Accordingly, logical file numbers 1 to 150 are used as file labels. The file routines are re-entrant, and hence may be called from both mainline and interrupt levels.

The file system routines perform all disk-storage allocation automatically, maintaining each file as a linked list of disk sectors. (A disk sector is the minimum addressable unit of data on the 2310; each contains 320 16-bit words.) The addresses of the first and last sectors of each file are kept in a separate table, called the Directory, also stored on the disk.

Initially, a file has no storage assigned to it and has no Directory entry; it is logically nonexistent. When it is "created," it is assigned one sector of storage from the top of a free-storage list and an appropriate entry is made in the Directory. If the file overflows that sector, it is linked to

the next sector on the free-storage list; additional sectors continue to be assigned in this way as long as necessary.

When a file is "destroyed," its Directory entry is zeroed and its sectors are returned to the top of the free-storage list. Thus, the first file to require a new sector following a destroy operation will be linked to the first sector of the destroyed file's storage.

As far as the user is concerned, the unit of data for the file system is the eight-bit byte. The PUTC routine must be called once for each byte to be placed into a file; the GETC routine returns one byte on each call. Since two words of each sector are required for link pointers, 318 words—636 bytes—are available to store data. The 636th consecutive call to the PUTC routine for a given file will cause a new sector to be linked to that file. The next call to PUTC will store a character in the newly linked sector's first data byte.

(GETW and PUTW, routines to fetch and store 16-bit words rather than 8-bit characters, will be made available in a future version of the file system as the need arises.)

It is not possible to write a partial sector on the 2310 disk (without loss of data), nor is it possible to begin reading at other than the beginning of a sector. In addition, every disk operation requires two control words adjacent to the actual data area in core storage. Consequently, a 322-word buffer area is required for reading or writing a file, and also for creating or destroying files (to permit updating the Directory,

in the latter cases). The user must provide this buffer area within his own program, and pass its address to the file routines as explained below.

Before a file can be read or written, it must be "opened." A file is "opened in " if it is to be read; this causes the first sector to be read into the user-supplied buffer area. If a file is to be written, it must be "opened out"; this causes the appropriate link pointers to be placed in the user's buffer. Both of these operations are performed by the routine OPEN.

In order to allow a single user to have an arbitrary number of files open at one time, and to facilitate re-entrant coding in the file routines themselves, all the information needed to access a file is contained in a five-word File Control Block (FCB), which will be established in a user-supplied area by the OPEN routine. By convention, Index Register 1 must always point to the first word of the file control block whenever a call is made to the file routines; violation of this rule may produce disastrous results.

When a file is opened, a bit is set in the FCB to indicate that condition; the GETC and PUTC routines always check this bit. Thus an attempt to read or write an unopened file will result in an error return.

In addition to being opened before use, files must also be closed after being used. For an output file, a call to CLOSE forces the writing of the last (partial) sector to the disk and also resets the "open" bit in the FCB. For input files, CLOSE

merely resets this same bit, thus the closing of input files is presently unimportant. However, a new version of the file routines (to be released soon) will incorporate interlocks to prevent the same file being opened more than once simultaneously, and correct operation of this interlock will require that all files be properly closed after use.

Once a file control block has been established by the OPEN routine, it should not be altered by the user. It contains such information as the address of the disk-buffer area for the file, the currently accessed sector address, a character pointer whose value is adjusted by the GETC and PUTC routines, some status information, and the logical file number. A user may open simultaneously as many files as desired, as long as all have separate disk-buffer areas and FCBs. Any given file, however, should not be opened both "in" and "out" at the same time, nor should it be opened multiply with separate FCBs at one time. (Neither of these latter two conditions is checked by the current version of the file routines, but both will cause error returns in the new version.)

In order to function correctly, the file routines require access to certain items of information (such as the total size of the disk file area) which are not available at assembly time and which cannot be obtained from the TSX core load builder. Consequently, there is a special initializing routine named WAKUP which reads some of the needed items from the disk and computes others, and which must be called (once) before any other calls are made to file system routines.

CALLING SEQUENCES

Abbreviations:

A = A-Register or Accumulator

Q = Q-Register

XRn = Index Register n

Error Returns: if k separate error returns are possible on a given call, the normal return is to CALL+k+1, while "error 1" returns to CALL+1, "error 2" to CALL+2, ..., "error k" to CALL+k.

All routines save and restore all registers except as otherwise noted.

1. To create a file:

   A = logical file number, right-adjusted

   Q = address of a 322-word disk buffer region

   XR1 = address of a 5-word block for FCB

   CALL CREAT

   Error Return - Illegal file number

   Error Return - No more room on disk

   Error Return - File already exists

   Normal Return

2. To destroy a file:

   A = logical file number, right-adjusted

   Q = address of 322-word disk buffer

   XR1 = address of 5-word block for FCB

   CALL DESTR

   Error Return - Nonexistent file or file open

   Normal Return

3. To open a file:

A_{8-15} = logical file number

A_0 = 0 for output file, 1 for input file

Q = address of 322-word disk buffer

XR1 = address of 5-word block for FCB

CALL OPEN

Error Return - Nonexistent file

Normal Return

4. To close a file:

XR1 = address of FCB

CALL CLOSE

Error Return - File not open or bad FCB

Normal Return

5. To put a character into a file:

XR1 = address of FCB

A = character to be inserted (right-adjusted)

CALL PUTC

Error Return - File not open

Error Return - No more room on disk

Normal Return

6. To get a character from a file:

   XR1 = address of FCB

   CALL GETC

   Error Return - File not open, etc.

   Error Return - End of file

   Normal Return - Character in A, right-adjusted

7. To initialize the file system routines:

   CALL WAKUP

   Normal Return - (Destroys A, XR1, XR2.)

3.2 Disk File Utility Program: User's Guide

The file utility program for the IBM 1800 is a keyboard-oriented command interpreter which allows the user to access the disk file system in a simple way and to perform a number of very useful, relatively deivce-independent, input-output operations. Two commands allow the user to create and destroy disk files; a third command allows a copying connection to be established between any pair of I/O devices on the system. (The structure of the disk file system is outlined in Section 3.1, which should be consulted for further information on the CREATE and DESTROY functions, and on the COPY function when one port is a disk file.)

The COPY operation is completely character-oriented; all I/O devices appear as single-character sources and/or sinks. Logical lines are delimited by carriage-return characters ("new-

line" in EBCDIC parlance), and fields within a line are determined by tabs, where applicable. Thus whenever data are read from an essentially line-oriented device such as the card reader, a format specification must be supplied to indicate where the tabstops are. Then the device-support routines for the card reader (or card punch, etc.) convert between the tab-cr character stream and the card-image line, as required. The advantages of this approach are greater economy of disk storage space for card-image data, faster operation of the IBM 1053 printer (because of the use of tabs), higher information density in dataphone records, and compatibility with the PDP-7 system.

The file utility program is designed to run under the TSX operating system for the 1800; the current version is set up as a "nonprocess" job. It calls approximately fifty subroutines.

The following is a description of commands recognized by the file utility. Each command may be abbreviated to the two underlined letters. Optional parameters are indicated in brackets ([ ]); parameters of which exactly one must be chosen are shown in braces ({ }).

CREATE xxx

where xxx = any valid file number (1 - 150_{10})

Action: Logical file xxx is created and allotted one sector of disk storage.

DESTROY xxx

where xxx = same as above

Action: Logical file xxx is destroyed and all of its storage released.

COPY [{xxx / dev1}] [{yyy / dev2} [z]]

where xxx = number of source file

dev1 = name of source device

yyy = number of sink file

dev2 = name of sink device

z = format parameter

xxx, yyy may be any valid (and existing) file number.

dev1, dev2 may be any of the following (and may be abbreviated by a single character):

TYPEWRITER	(keyboard if source, printer if sink)
CARD	(reader if source, punch if sink)
PDP-7	(source or sink)

z may be any of the following:

- ASM - 1800 Assembler tab stops (col. 27, 32, 35, 45, starting in col. 21)
- PDP-7 - PDP-7 format tab stops (col.10, 25, 40, 55, 70, starting in col. 1)
- FORTRAN - FORTRAN tab stops (col. 7, starting in col. 1)
- DUMP - "dump" format; full 80-column card, no tab stops
- BINARY - 1800 binary format, one 8-bit byte per column, 80 columns per card

(In the first four formats, all characters are punched or read in EBCDIC.)

Action: Characters are copied from the source file or device to the sink file or device until an end-of-file condition exists at the source.

End-of-file conditions are logical end-of-file return from the file system's GETC routine, or detection of an EOB character (card-code 0-6-9, EBCDIC 26_{16} , ASCII 203_8) by the card reader or PDP-7 port. A copy operation may be aborted at any time by turning data switch 0 on.

Default Parameters: dev1 = CARD, dev2 = TYPEWRITER,
z = ASM

LIST [{ xxx / dev1 }]

xxx, dev1 same as for COPY

Action: Characters from the source file or device are printed on the 1053 printer in 1800 assembler format. (This command is actually identical to COPY, and is included mainly for historical and aesthetic reasons.)

DEBUG

Action: Control is transferred to the 1800 debugging routine, which is described in Section 4. This feature has been included mainly for its utility in debugging this and other system programs, particularly where disk patching is required.

BLAST

Action: Clear the pending "read" on Digital Input, thereby disabling all PDP-7 interrupt service.

PDP7

Action: Reinitialize PDP-7 interrupt service.

TAB $\left[\left\{\begin{matrix}A\\P\end{matrix}\right\}\right]$

Action: Spaces the 1053 printer across the page, pausing for approximately one second at each 1800 assembler ("A") or PDP-7 ("P") tab stop to allow the operator to set the typewriter tab stops correctly. (The default parameter is "A".)

EXIT

Action: Return to TSX supervisor, which will begin reading cards looking for // JOB.

Two additional commands, FIX (to reset file system interlocks in event of user-program malfunction or system snark) and SNIFF (to compute and print storage-use statistics and other information about existing files) will be available in the near future.

3.3 PDP-7 Interrupt Service: "PDP7"

The PDP-7 interrupt service subroutine, "PDP7," is a command-dispatching program which runs on the 1800, enabling the latter to perform certain services for the PDP-7, upon its request. Each command issued by the PDP-7 causes an interrupt at the 1800 which transfers control to the "PDP7" program; the service function is then performed "at the interrupt level" (in IBM terminology), after which the 1800 resumes execution of the interrupted program.

The PDP7-1800 interface currently in use allows single words to be transferred in parallel in either direction—12-bit words when sending from the 1800, 16-bit words when sending from the PDP-7. By convention, the basic unit of data is the 8-bit byte in the low-order portion of the word. The additional 8 bits are set by the PDP-7 to indicate which function is desired, while the additional 4 bits available to the 1800 are used as an acknowledgment code.

The 1800 prepares to receive commands from the PDP-7 by establishing a one-word "read" on Digital Input.* When it has finished performing a function for the PDP-7, the 1800 reestablished this "read." Thus, whenever the PDP-7 presents a word to the 1800, it completes the pending I/O operation and causes an interrupt.

The interrupt service subroutine performs most of its functions by calling other subroutines, including the disk file system. Sufficient buffer space is included in "PDP7" to maintain two disk files open at once; by convention, one is an input file and the other output. Thus, the PDP-7 may simultaneously read and write disk files. Since file creation and/or destruction also requires buffer space (see disk file system description), a small buffer management routine saves one of the file buffers on the disk when necessary, thereby allowing files to be created and destroyed without regard for the state of files

* See IBM 1800 Functional Characteristics. IBM Form A26-5918.

currently being read or written. Legal commands to "PDP7," and their corresponding functions, are listed in Table 1. The 1800 indicates positive acknowledgment (command accepted and function performed) by inserting 1010_2 into the high-order bits of its response word; negative acknowledgment is indicated by 1100_2 in those bits, with an error code (see Table 2) in the data byte. In the present system, every word sent by the PDP-7 is a command, and every command is acknowledged by the 1800. (Most of the commands recognized by "PDP7" involve the disk file system (see Section 3.1), whose description should be consulted for further clarification of these functions.)

An exception here is the Send File command, which is intended for use in loading PDP-7 system programs from the 1800 disk. Receipt of this command causes the 1800 to transmit the entire contents of the designated file to the PDP-7 without intervening acknowledgments; the 1800 merely sends each character immediately after the PDP-7 accepts the previous one. If the PDP-7 does not read the entire file as expected, the 1800 is left in a permanent wait loop at an "interrupt level" which can be terminated only be operator intervention. Hence the Send File command is rejected if it does not reference a system file.

The "1800 copy port" is a pair of reserved words, IN and OUT, in 1800 memory through which other programs can communicate with the PDP-7 interrupt service subroutine, "PDP7." The low-order byte of each of these words is used for data, the high-order byte for control. An 1800 user program sends data to "PDP7"

(i.e., the interrupt service subroutine, not the PDP-7 computer) by storing the data byte in OUT with the control byte (the high-order byte) nonzero. "PDP7" interprets a nonzero control byte to mean "data present"; the first subsequent "read" command from the PDP-7 will cause the data byte to be transmitted to the PDP-7. "PDP7" then zeroes OUT to signal the user program that the next data byte may be sent. All further "read" commands from the PDP-7 are rejected until the control byte becomes nonzero again, indicating that the user program has stored another data byte in OUT.

The same convention applies to transmission in the other direction. The user program indicates readiness to accept another data byte by zeroing IN; the first subsequent "write" command received from the PDP-7 causes the data byte to be stored in IN with the control byte nonzero. Until the control byte is made zero again by the user program, "write" commands from the PDP-7 are rejected.

Since the file utility program (see Section 3.2) can reference the 1800 copy port, it is possible to copy binary or character data from any device on the 1800 to or from the PDP-7 or any of its devices, including in particular the display and the dataphone.

Table 1. "PDP7" Commands and Services

Command	Code (Decimal)	Code (Octal)	Code (Hex)	Function
Echo	1	1	1	Write data byte back to PDP-7 (used for checking).
Open Out (translated)	2	2	2	Open an output file (number specified by data); ASCII-EBCDIC translation on.
Open In (translated)	3	3	3	Open an input file, as above.
Put	4	4	4	Place character in current output file.
Get	5	5	5	Fetch character from current input file.
Close Out	6	6	6	Close current output file.
Close In	7	7	7	Close current input file.
Clear	8	10	8	Close all current files.
Create	9	11	9	Create file n (data byte=n)
Destroy	11	13	B	Destroy file n (data byte=n)
Open Out (binary)	13	15	D	Open output file n (data byte=n); ASCII-EBCDIC translation off.
Open In (binary)	14	16	E	Open input file n (data byte=n); ASCII-EBCDIC translation off.
Echo A	16	20	10	Translate EBCDIC to ASCII
Echo B	17	21	11	Translate ASCII to EBCDIC

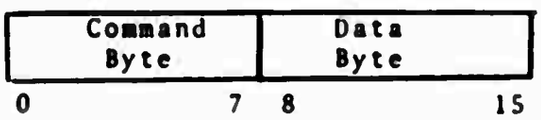
Table 1. continued

Command	Code (Decimal)	Code (Octal)	Code (Hex)	Function
Send File	18	22	12	Write entire file n (data byte=n) to PDP-7.
Read	19	23	13	Read character from 1800 copy port (EBCDIC-ASCII translation on).
Write	20	24	14	Write character to 1800 copy port (ASCII-EBCDIC translation on).
Read Binary	21	25	15	Read byte from 1800 copy port (EBCDIC-ASCII translation off).
Write Binary	22	26	16	Write byte to 1800 copy port (ASCII-EBCDIC translation off).

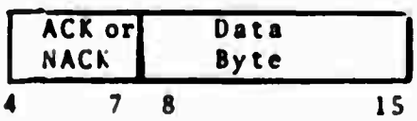
Table 2. "PDP7" Error Codes

Code (Decimal)	Code (Octal)	Code (Hex)	Meaning
1	1	1	Invalid command.
2	2	2	"Open" request rejected (no such file, etc.)
3	3	3	"Open" request rejected (file already in use)
4	4	4	"Put" error (file not open)
5	5	5	File overflow (no more room on disk-during "Put" attempt)
6	6	6	"Get" error (file not open)
7	7	7	"Close" error (file not open)
8	10	8	"Create" error (illegal number)
9	11	9	"Create" error (no more disk room)
10	12	A	"Destroy" error (no such file exists)
11	13	B	EBCDIC-ASCII translation not possible
12	14	C	ASCII-EBCDIC translation not possible
13	15	D	Copy port busy-try again
14	16	E	Invalid number (i.e., not a system file) in Send File command.

PDP-7 Command format:



1800 Response format:



4. DEBUG: A KEYBOARD DEBUGGING PACKAGE FOR IBM 1800

4.1 Introduction

DEBUG is a program checkout aid for the IBM 1800, written in June 1968, as a temporary measure, but still in use. It allows examination and modification of registers and core locations, storage-protection and -unprotection of core locations, transfer of program control, copying of core from and to the disk storage, and comparison of core locations with their previously saved contents. A trace facility was also included, but works unreliably due to bothersome aspects of the interaction with the operating system; no further mention is made here of the trace feature.

Programmer and machine communicate through the typewriter-keyboard. The user program must execute a CALL DEBUG statement to establish linkages. DEBUG types out an entry message, and the user may begin typing commands. After this, DEBUG may be reached by various means from the user program; on entry, it explains to the user why his program stopped executing:

DEBUG typeout	Reason for Entry
CAL:	User program called DEBUG
CI:	User pressed Console Interrupt button
STOR PROT:	User program committed Storage Protection error.
OP CODE:	User program committed Op Code error
PARITY:	Parity check occurred
CAR CHECK:	Channel Address Register Check occurred.

After this explanatory typeout, DEBUG types its entry point to tell the user where his program stopped. (As usual, the instruction previous to the entry point value was the last instruction executed.)

4.2 Arguments

All data input and output through DEBUG are four-digit hexadecimal numbers, either absolute numbers or displacements relative to a BASE value. There are no symbolic values. Absolute numbers are prefixed with a period, relative values with an asterisk.

Examples:

.1000eof	absolute 1000(hex)
*1000eof	BASE plus 1000
*eof	BASE plus zero
.eof	zero
eof	zero

(eof is the end-of-file key.)

Escape to command mode is accomplished by entering pound-sign # as the first character of an argument; #eof is sufficient.

DEBUG signals its readiness for an argument input by typing a left parenthesis. If the argument entered doesn't begin with '#', '*', or ".", DEBUG types '?' and accepts a new argument; otherwise it types a right parenthesis.

4.3 Registers and Commands

DEBUG uses a small set of core registers to control its operations; the effect of various commands is most concisely described by their effect on these registers; see Table 3. The sample session (Section 4.4) would make clear the use of the registers and commands.

Table 3. DEBUG Registers and Commands

Registers

BASE	Base for relative arguments
LOC	Core location currently referenced
SABUF	Buffer for registers saved on entry; A, Q, X1, X2, X3, Status
LOWE,HIE FILO, FIHI	Four sequential locations used to record core limits. LOWE and HIE aren't used any longer; FILO is the low end of core written in a core-save ('F') operation and read by a compare ('C') operation; FIHI is the high end. The defaults (.1DE0,.3FFF) should be satisfactory for most uses.
SNPMO	The mode (absolute or relative) in which LOC is typed out
DAREA, LENG ABSRL, SECT	Four sequential locations controlling disk input-output. DAREA is core address of first word of data written or read; two preceding words are saved and restored during the operation. LENG is the length of core buffer read or written if it is less than DAREA or if it has a /8000 bit added; otherwise it is the core address of the high end of the buffer. ABSRL is the mode bit for the disk operation; 0 for absolute sector address, 1 for sector address relative to Non-Process Working Storage SECT is the sector address

Commands

(When a command is followed by parentheses( ), it means that it accepts one argument, denoted here by ARG.)

B( )	ARG → BASE
S( )	ARG → LOC, set SNPMO = ABS or REL
.	Type out I LOC absolute

Table 3. Continued

,	Increment LOC by one
<	Decrement LOC by one
;	Increment LOC by one, return carriage, type out LOC in mode of SNPMO
M( )	Mode of ARG → SNPMO. E.g., M(.), or M(*).
L	Type out LOC in mode given by SNPMO.
=( )	ARG → 1 LOC. If 1 LOC is store-protected, make the change and type out '!', leaving the location store-protected
P	Address of LOWE → LOC. Allows examination and change of limits.
A	Address of SABUF → LOC. Allows examination and change of saved registers.
J( )	Unsave registers from SABUF. BSI 1 ARG. ARG must be non-zero. On return from subroutine, the saved registers in SABUF are not altered from their previous values.
R( )	Unsave registers from SABUF. If ARG ≠ 0, BSC I ARG. If ARG = 0, BSC 1 ENTRY. I.e., return via DEBUG entry point.
!	Store protect 1 LOC
\	Unprotect 1 LOC
D	Address of DAREA → LOC, allowing examination and change of disk I/O parameters.
K( )	ARG → SECT. Read disk (absolute or relative sector address determined by ABSRL), beginning at sector SECT, storing into 1 DAREA to I LENG or from 1 DAREA to (1 DAREA) + LENG. (See explanation of LENG in the registers above.)
W( )	ARG → SECT. Write disk with the same parameters as in K( ) above, mutatis mutandis.

Table 3. Continued

F( )	ARG → SECT. Dump two blocks of core onto disk. First block is from I FILO to beginning of DEBUG core; it goes into relative sector SECT in NPWS. Second block is from end of DEBUG core to I FIHI; it goes onto the sectors following the first block.
C( )	ARG → SECT. Read the two blocks (written by F( ) presumably) sector by sector, comparing each word to the corresponding contents of core. Whenever there is a difference, type out its location and the number of consecutive words which are different from their previous values. Except that if an isolated word has changed, type '=' and the contents of the word.
E( )	Exit to TSX. This is necessary to restore several low-core values; the user should always terminate his run with E( ) . The ARG isn't used for anything, but gives the user a chance to escape back to command mode if typing 'E' was a mistake.

4.4 Sample Session

```
// ASM TEMP
*LIST
0000 30 04142907      BEGIN CALL   DEBUG
0002 01 C400000A            LD    L A
0004 01 8400000B            A     L B
0006 00 D4000008            STO   L 8        ERROR, STORE-PROTECTED CORE
0008 0  0000                DC      0        ERROR, INVALID OP CODE
0009 0  70F6                MDX     BEGIN
000A 0  0001          A     DC      1
000B 0  0004          B     DC      4
000C 0  0000          C     DC      0
000E    0000                END     BEGIN
    NO ERRORS IN ABOVE ASSEMBLY.
TEMP
DUP FUNCTION COMPLETED
 // XEQ TEMP
 *CCEND

CLB, BUILD TEMP

CLB, TEMP  LD XQ
CAL:.1DE2  [Debug was called, .1DE2 is return address.]

B(.1DE0)   [Set BASE to the beginning of sample program.]

S(*000A) .0001 , .0004 , .0000  [Look at the data words.]

R()

STOR PROT:*0008  [Catches the store protect error.]

S(*0007) .0008 (*000C) R(*0002)  [Change the address to *000C,
                                  start over]

OP CODE:*0009  [Now op code check comes up]

S(*0008) .0000 = (.1000) R(*0002)  [Change the zero to a NOP,
                                    start over]

CAL:*0002  [Went all the way back to top]

S(*000C) .0005 = (.0000) R(#  [Data look good, set back to zero.

F() E()                       Started to go back to program,

CAL:*0002                     then decided to save core on disk]

C()      [Perform the compare operation]
```

1DEC =0005

E() [Exit back to TSX]

NO4 READY READER

Underlined typeout was typed by DEBUG.
Carriage returns are ad lib; the Carriage Return key doesn't communicate with the 1800.
Arguments are terminated by an EOF character, but the EOF doesn't print on the typewriter.

5. DISK ASSEMBLER FOR IBM 1800

5.1 Reading Source Lines from Disk Using IBM 1800 TSX Assembler

The TSX Assembler is a two-pass assembly. On its first pass, it normally reads card input, creates a symbol table, and writes a packed-format intermediate disk output. On the second pass, each line of the intermediate output is read back in, unpacked back into card format, and processed. (See Figure 1, Normal Assembler Operation.)

Because of the strict modularity of this assembler, we were able to modify it so that both passes read from the disk. To assemble a program, we must first copy it onto the disk in the format which the assembler reads. If the source program is on cards, the disk-copy is accomplished by the program CDISK; if the source is in a character file (see Section 3, 1800 Logical File System), the copying is done by FDISK (see Figure 2, Modified Assembly Procedure).

5.2 Literal Constants

The programs FDISK and CDISK include a routine called LITS, a processor for literal constants. LITS replaces the characters of a literal (e.g., '8') with a generated name (e.g., LT001), and saves the literal text in an accumulation table. When an LTORG line is encountered, the accumulated literals are written out as regular program lines, e.g., LT001 DC 8. Thus, the literal processor reads and writes character lines; the assembler itself has no capacity to process literal constants.

Figure 3 shows how LITS works within CDISK; its role in FDISK is analogous.

5.3 The Format of Literal Constants

Four main types of literals may be used in the Operand field of an instruction:

1. Data Type. Generates a DC with literal text operand.
2. EBC Type. Generates an EBC with literal text operand.
3. DMES Type. Generates a DMES with literal text operand.
4. Instruction Type. Generates an instruction. Literal text is replaced into col. 27 onwards, with the BAR character '|' indicating a Tab to next field.

These are written as follows:

1. Data Type. ='Text' or more simply 'Text' (Equal sign may be omitted.)
2. EBC Type. =E'Text' where Text should have periods at each end as required by the EBC pseudo-op of the assembler
3. DMES Typs. =D'Text'
4. Instruction Type. =I'OP|Tag|Operand|Comments'

A literal may be used anywhere in the operand field preceding the first blank, just as any other symbol. The literal text is delineated on the right by scanning from the right for the right-most prime. Therefore, primes should not be used in comment fields.

The pseudo-op LTORG causes the literals accumulated since the beginning of the program (or since the last LTORG) to be written out. (In a long program it is advantageous to use several LTORGs so that literals may be kept within short-instruction range.) The END line does not dump literals; an explicit LTORG must be supplied.

Two literals with identical text will generate only one literal constant (if they are not in different LTORG sections). Two literals with different text will always generate different constants even if the assembler may evaluate their text identically (e.g., '8' and '/8' will generate different constants).

Figures 4 and 5 illustrate the use of literals.

5.4 How to Use CDISK, FDISK, and the Modified Assembler

CDISK: Place your card deck in the following sequence:

```
//JOB
//XEQ CDISK   FX
...Your assembly deck, terminated with END card...
//ASM name                    ]
*LIST                         |
*PUNCH                        |  Optional assembly control cards
*SYSTEM SYMBOL TABLE          |
etc.                          ]
*STORE                        ]  Optional DUP control cards
*STOREMD                      ]
//END                         ]  Post-assembly options
//XEQ                         ]
```

FDISK:

The source program must be in a disk character-file, numbered NNN say. The assembly card deck should be as follows:

```
//JOB
//XEQ FDISK  FX
bbbNNN                    Your decimal file number, right-
//ASM name                justified in the first six columns
*LIST
*PUNCH                    Optional assembly control cards
*SYSTEM SYMBOL TABLE
 etc.
*                         special star-in-column-21 card
*STORE
*STOREMD                  Optional DUP control cards
//END
//XEQ                     Post-assembly options
```

Literal constant listing:

If, and only if Data Switch 2 is OFF, literal constants are listed as they are encountered. Each LTORG causes a blank line in the listing. Error procedures:

Error	Effect	Correction procedure
END line missing	CDISK: '105 RESTART' typeout FDISK: 'NO END CARD' typeout	CDISK: Insert END card, assemble again FDISK: Abort job*, insert END in file, assemble again.
Format error in literal	'FORMAT ERROR' typeout	Abort job,* fix the format error, assembler again.
LTORG line missing	Assembler reports undefined symbols	Insert LTORG, assemble again.

* 'Abort job' means 'Press console interrupt with Sense Switch 7 ON.'

Literal format errors:

a. The next character after an = is not ', D, E, or I.

b. The right-hand ' is omitted; only one ' in this operand.

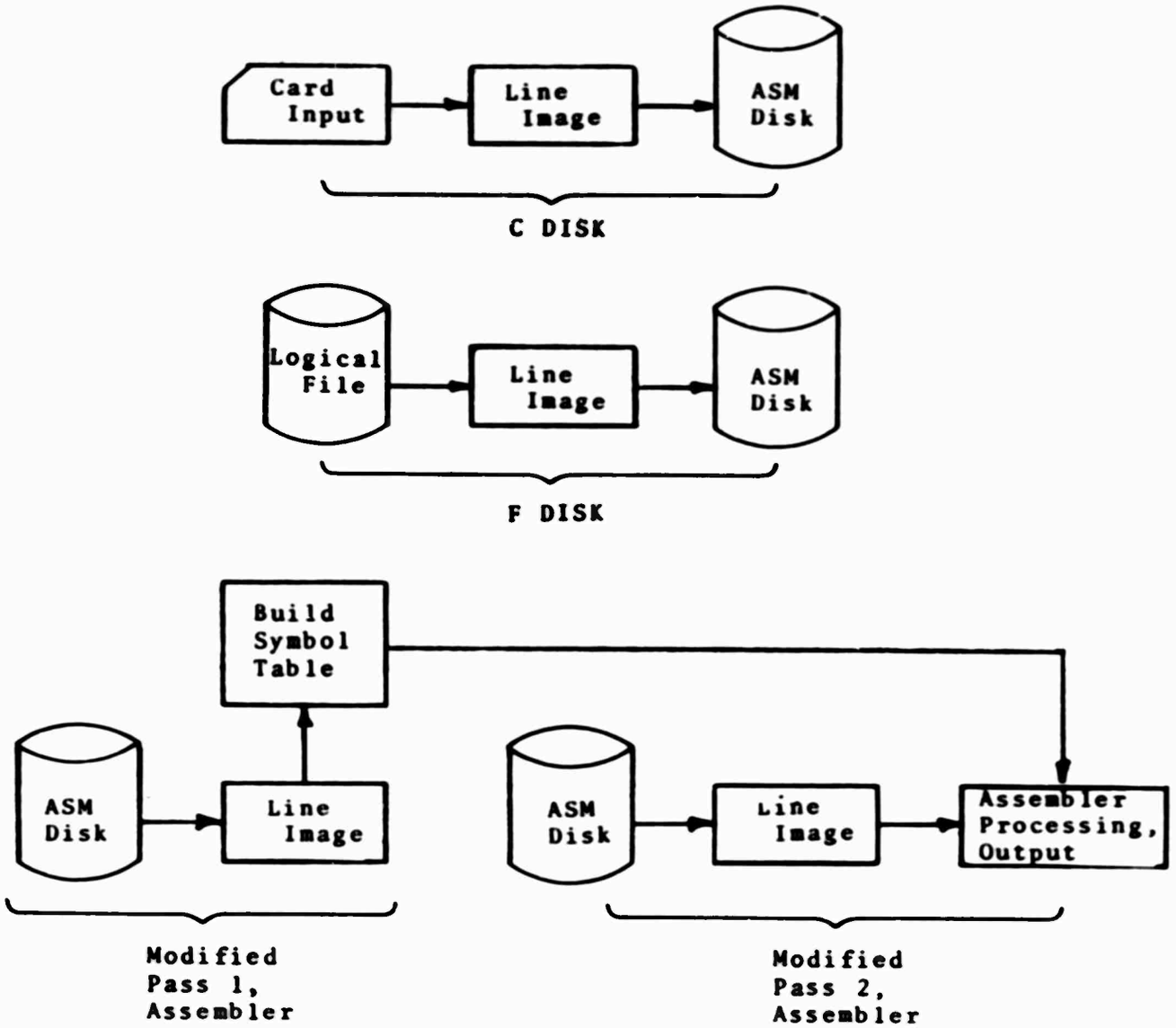


Figure 2. Modified Assembly Operation

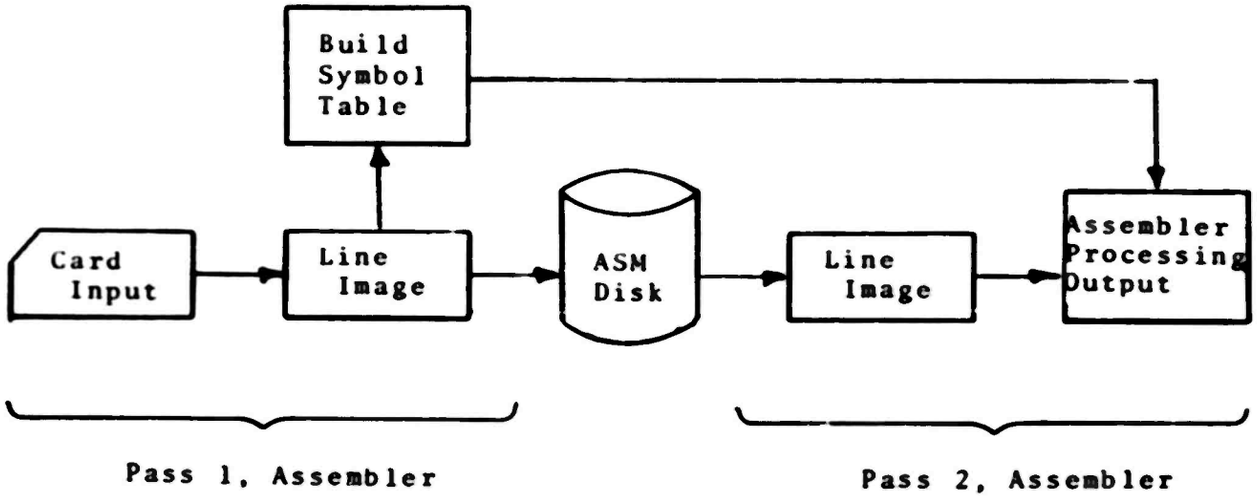


Figure 1. Normal Assembly Operation

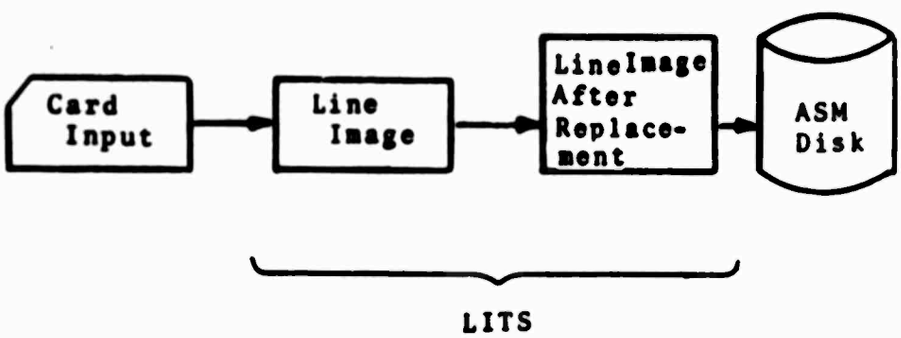


Figure 3. Operation of LITS within C DISK

```
* EXAMPLE OF 1800 LITERAL PROCESSING
*
RETUR NOP
DEMO  LD      '100'       ABBREVIATED FORMAT
      A       =F'200'     FULL FORMAT
      A       '/8000'     HEX LITERAL
      M       '.A'        CHARACTER LITERAL
      LD   L  'RETUR'     ADDRESS CONSTANT
      LDX  L1 BASE-'100'
      LTORG               LTORG DUMPS LITERALS HERE
BASE  EQU     *
*
      DC      =E'.CHARS.'           EBC TYPE LITERAL
* NOTE THAT EBC LITS NEED PERIODS IN IT
      LD      =D''RHELLO'E'         DMES LITERAL
      LD      =I'NOP|||SWITCH'      INSTRUCTION LITS
      LD      =I'WAIT'
      LD   L  =I'BSC|I|RETUR'
      LTORG               DON'T FORGET
      END     DEMO
```

Figure 4. Example of Literal Usage: Source Listing

```
 // JOB
 // XEQ CDISK   FX
LT001 DC       100
LT002 DC       200
LT003 DC       /8000
LT004 DC       .A
LT005 DC       RETUR

LT006 EBC      .CHARS.
LT007 DMES     'RHELLO'E
LT008 NOP                SWITCH
LT009 WAIT
LT010 BSC  I   RETUR

 // ASM DEMO
 *LIST
                       * EXAMPLE OF 1800 LITERAL PROCESSING
                       *
 0000 0  1000          RETUR NOP
 0001 0  C007          DEMO  LD       LT001     ABBREVIATED FORMAT
 0002 0  8007                A        LT002    FULL FORMAT
 0003 0  8007                A        LT003   HEX LITERAL
 0004 0  A007                M        LT004      CHARACTER LITERAL
 0005 01 C400000D            LD   L   LT005   ADDRESS CONSTANT
 0007 00 65000005            LDX  L1  BASE-LT001
                       *     LTORG             LTORG DUMPS LITERALS HERE
 0009 0  0064          LT001 DC       100
 000A 0  00C8          LT002 DC       200
 000B 0  8000          LT003 DC       /8000
 000C 0  00C1          LT004 DC       .A
 000D 1  0000          LT005 DC       RETUR
 000E                  BASE  EQU      *
                       *
 000E 1  0014                DC       LT006          EBC TYPE LITERAL
                       * NOTE THAT EBC LITS NEED PERIODS IN IT
 000F 0  C007                LD       LT007        DMES LITERAL
 0010 0  C009                LD       LT008    INSTRUCTION LITS
 0011 0  C009                LD       LT009
 0012 01 C400001C            LD   L   LT010
                       *     LTORG              DON'T FORGET
 0014    0005          LT006 EBC      .CHARS.
 0017    0006          LT007 DMES     'RHELLO'E
 001A 0  1000          LT008 NOP                SWITCH
 001B 0  3000          LT009 WAIT
 001C 01 4C800000      LT010 BSC  I   RETUR
 001E    0001                END      DEMO
     NO ERRORS IN ABOVE ASSEMBLY.
DEMO
DUP FUNCTION COMPLETED
 // END
```

Figure 5. Example of Literal Usage: Assembly Listing.

6. 1800-PDP-7 INTERFACE

6.1 The "Minor" 1800-PDP-7 Interface

This interface provides full-duplex single-character transfer capabilities between the IBM 1800 and the PDP-7 under direct program control. Data transfer rates of approximately 1000 8-bit characters per second make it the fastest I/O device currently (September 1968) on the PDP-7. This section describes the interface as it currently functions, but does not describe facilities physically present but non-functional.

(This interface is the patched-up remains of an interface intended to provide block transfer capabilities but which was never completed. The current capabilities were completed to provide some interface until a suitable new design could be provided and built. Heavy use has been made of even this limited interface.)

The interface consists of an 1800 digital register output and digital voltage sense input I/O options connected to the 1800 via a data channel, and some additional hardware built specifically for this application. The latter contains two registers called(for historical reasons), TBUF and COUNT, and three PDP-7 program-accessible flags used for control purposes.

For use of the interface from the 1800, the reader is referred to the bibliography IBM 1800: Functional Characteristics. The following notes are offered in addition. Both digital-in and digital-out use the "external sync" options to synchronize

with the remainder of the interface. The digital-in must be initialized before any input can be accepted from the PDP-7. On digital-out, the operation-complete interrupt will occur when the last word has been transferred from 1800 core memory to the digital-out register. However, the digital-out device becomes "not busy" only after the external device has accepted the data from the digital-out register and returned the sync signal. Thus the 1800 must use the digital-out device status word (DSW) to determine if an output operation is really complete.

FLAGS

Three flags are provided to the PDP-7 program for control purposes. The REQDN718 indicates that a word has been transmitted to the 1800. The INTRQ18 indicates that the 1800 has a word for the PDP-7. The INTENBL flag enables REQDN718 or INTRQ18 to make an interrupt request to the PDP-7. If interrupts are on, then an interrupt will result.

The INTENBL flag is set by the SET command. The flags are cleared by a CLR command where the flag(s) cleared are specified by bits in the accumulator. Bit 0 refers to INTENBL, Bit 1 to INTR18, and Bit 2 to REQDN718.

Two skip instructions are provided to allow testing the state of the flags. SKIP718 will cause a skip if the REQDN7.8 flag is set. SKIP187 will cause a skip if the INTRQ18 flag is set.

Note that the PDP-7 Clear All Flags (CAF = IOT 3302) will also clear all three flags, as will pressing START on the PDP-7 control panel.

TRANSFERS FROM PDP-7 TO 1800

The 16-bit TBUF register is loaded from the low-order 16 bits of the PDP-7 accumulator by the LDTBUF command. When this word has been loaded and the 1800 has a read (with external sync) pending on the digital-in device, then that word is transferred from TBUF to 1800 core as specified by the digital-in data channel. If that transfer satisfies the word count for the channel, then the 1800 may be interrupted by the operation-complete state of the data channel. In practice, the 1800 must always maintain an outstanding read on the digital-in in order to be responsive to PDP-7 action.

The transfer of the data from the TBUF to 1800 core causes the REQDN718 flag to set, indicating that the requested transfer has taken place. The state of this flag may be sensed as explained above.

TRANSFERS FROM 1800 TO PDP-7

Transfers from the 1800 to the PDP-7 are initiated by the 1800 writing to the digital-out device. The digital-out indicates by an external sync signal that it has data to be transferred. If the INTRQ718 flag is not set, then those data are transferred into the 12-bit (yes, twelve) COUNT register, and the INTRQ18 flag set indicating data are present. This flag

may cause an interrupt or be sensed. The data are read into the accumulator of the PDP-7 in ones complement form by the RDDATA instruction. The INTRQ18 flag may then be cleared, thereby enabling the next word to be transferred from the 1800 to the COUNT register.

INSTRUCTION ASSIGNMENTS

The control instructions for this interface have the following encodings:

LDTBUF	=	IOT 2744
RDDATA	=	IOT 2751
SKIP718	=	IOT 2702
SKIP187	=	IOT 2722
CLR	=	IOT 2701
SET	=	IOT 2704

BIBLIOGRAPHY

IBM 1800: Functional Characteristics, Form A26-5918.

IBM 1800 Timesharing Executive System: Concepts and Techniques, Form C26-3703.

PDP-7 User's Handbook, DEC No. F-75, Digital Equipment Corporation, Maynard, Mass., 1965.

Brown, P.J., ML-1 User's Manual, available from DECUS, DEC User's Society; summarized in Communications of the ACM, Vol. 10, No. 10, October 1967, pp. 618-623.

PDP-7 Symbolic Assembler Programming Manual, DEC No, 7-3-S, Digital Equipment Corporation, Maynard, Mass., 1965.

Brender, R.F., Use of DDT with "Interrupts-On" Programs, Technical Memorandum, Concomp Project, University of Michigan, Ann Arbor, July 1967, 9 pp.

MACRO-9 Assembler: Programmer's Reference Manual, DEC-9A-AM9A-D, Digital Equipment Corporation, Maynard, Mass., 1967.

Frantz, D.R., Brender, R.F., and Foy, J.L. Jr., LOCOSS: A Multiprogramming Monitor for the DEC PDP-7, Technical Report 10, Concomp Project, University of Michigan, Ann Arbor, October 1968.

Brender, R.F., and Foy, J.L. Jr., Flexible High-Speed Interface between IBM 1800 and DEC PDP-7 Computers, Technical Report 12, Concomp Project, University of Michigan, Ann Arbor, October 1968.

Deutsch, L.P., and Lampson, B.W., "An Online Editor," Communications of the ACM, Vol. 10, No. 12, December 1967, pp. 793-799.

PDP-7 Symbolic Tape Editor Programming Manual, Digital 7-1-S, Digital Equipment Corporation, Maynard, Mass., 1965.

DOCUMENT CONTROL DATA · R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1 ORIGINATING ACTIVITY (Corporate author)
THE UNIVERSITY OF MICHIGAN
CONCOMP PROJECT

2a. REPORT SECURITY CLASSIFICATION
Unclassified

2b GROUP

3 REPORT TITLE
SPECIALIZED SYSTEM SOFTWARE FOR INTERACTING DEC PDP-7 AND IBM 1800 COMPUTERS

4 DESCRIPTIVE NOTES (Type of report and inclusive dates)
Technical Report 11

5 AUTHOR(S) (First name, middle initial, last name)
R.F. Brender, D.R. Frantz, J.L. Foy, Jr., and T.W. Schunior

6 REPORT DATE
December 1968

7a. TOTAL NO OF PAGES
-93-

7b. NO OF REFS

8a CONTRACT OR GRANT NO
DA-49-083 OSA-3050

b PROJECT NO

c

d

9a. ORIGINATOR'S REPORT NUMBER(S)
Technical Report 11

9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)

10 DISTRIBUTION STATEMENT
Qualified requesters may obtain copies of this report from DDC.

11 SUPPLEMENTARY NOTES

12 SPONSORING MILITARY ACTIVITY
Advanced Research Projects Agency

13 ABSTRACT

A collection of programs written for interacting DEC PDP-7 and IBM 1800 is described. These programs provide:

1. device support for interaction between 1800 and PDP-7,
2. a serial-by-character logical file system on the 1800 disk (2310) for use by both computers,
3. a file manipulation utility package,
4. a file-oriented text editor running on PDP-7 used for preparing both PDP-7 and 1800 programs,
5. modifications to the assemblers of each computer to read from the logical file system, and
6. a keyboard-oriented debugging package for the 1800.

A (temporary) single-character full-duplex interface between 1800 and PDP-7 is also described.

DD FORM NO. 1473

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
DEC PDP-7						
IBM 1800						
Interactive Computers						
File Systems						
Keyboard Debugging						
Multiprocessing						
Text Editing						